msdn® training

Module 4: Separating Code from Content

Contents

Microsoft®

Overview

- Advantages of Partitioning an ASP.NET Page
- Creating and Using Code-Behind Pages
- Creating and Using User Controls
- Creating and Using Components

Active Server Pages (ASP) applications contain a mix of Hypertext Markup Language (HTML) and script, making the code difficult to read, debug, and maintain. Microsoft® ASP.NET eliminates this problem by promoting the separation of code and content. That is, in ASP.NET, the user interface and the user interface programming logic need not necessarily be written in a single page.

There are three ways in which you can separate code and content in ASP.NET:

- By using code-behind files that are pre-compiled modules written in any of the Microsoft .NET run-time-compliant languages.
- By creating user controls—frequently used control sets and their logic—and using them as you would use controls in your ASP.NET pages.
- By moving business logic into components that can run on the server and calling those components from server-side code.

After completing this module, you will be able to:

- Explain the need for code-behind pages.
- Create a code-behind page and use it with an ASP.NET page.
- Explain the advantages of user controls.
- Explain how user controls work.
- Create a component in Microsoft Visual Basic®.
- Use a component in an ASP.NET page.

Advantages of Partitioning an ASP.NET Page

- **Individual Members of the Development Team Can Work on Separate, Individually-Owned Parts**
- **Developers Can Work Within Environments That Are Familiar**
- **Web Authors Can Use HTML Development Tools to Build the Interface**

Partitioning ASP.NET pages into code and content has several advantages. Partitioning eliminates confusing pages where code and HTML are intertwined, producing pages that are easier to maintain and understand. In addition:

- Members of the development team can work on their own parts without disturbing the work of others. For example, the interface designer can work on the interface files at the same time that the programmer works on the source code files.
- Partitioning code and content allows developers to use environments that are familiar to them. For example, you can use separate editors to develop the code.
- Web authors can use other HTML development tools to build the visible interface part of an application.

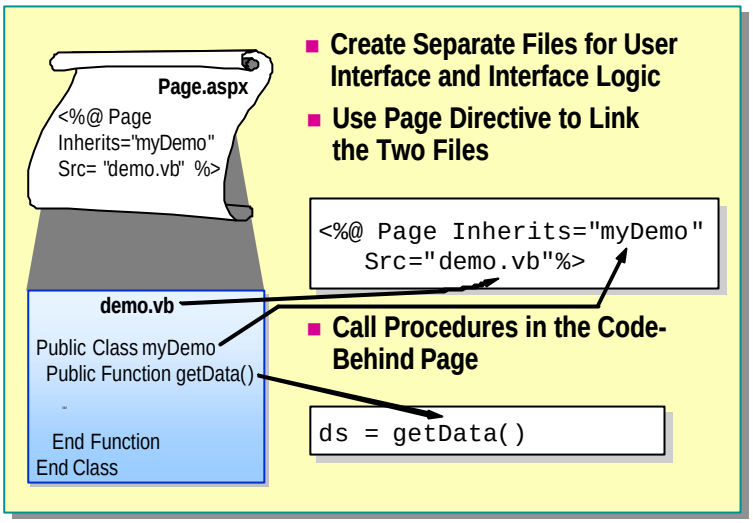
Creating and Using Code-Behind Pages

- Understanding How Code-Behind Pages Work
- Creating a Class File in Visual Basic
- Demonstration: Creating a Code-Behind Page
- Accessing Controls in a Code-Behind Page

User interface logic for a Web form relies on code that you create to interact with the form. You can design your application so that the code resides in a separate file known as the code-behind page, that is written in Visual Basic or Microsoft Visual C#™. When you run the Web form, the code-behind class file runs and dynamically produces the output for your page.

In this section, you will learn how code-behind pages work. You will also learn how to create a code-behind class file in Visual Basic.

Understanding How Code-Behind Pages Work



Creating code-behind pages is relatively simple. It involves creating two separate files, one for the user interface and the other for the user interface logic. The interconnectivity between these two files is provided through the @Page directive, which is used to specify optional settings at the page level.

Creating a User Interface File

First, you create the HTML for the interface. The interface file is an ASP.NET page (that uses the .aspx extension). The first line of the interface ASP.NET page is an @Page directive that specifies the name of the code-behind file and the actual class name inside the code-behind file that will be used.

In the following example, the ASP.NET page is linked to the demo.vb class file. The **Inherits** attribute specifies the class file to be used and the **Src** attribute indicates the path to the class file itself.

```
<%@ Page Language="vb" Inherits="myDemo" Src="demo.vb" %>
```

Note Specifying the path to the class file is optional. If the path is omitted, ASP.NET looks for the class file in the /bin directory of the application.

Creating a Code-Behind File

Next, you create a separate class file, in any of the supported languages, that provides the functionality required for the user interface page. A code-behind class file is identical to any other class file that you create in a particular language.

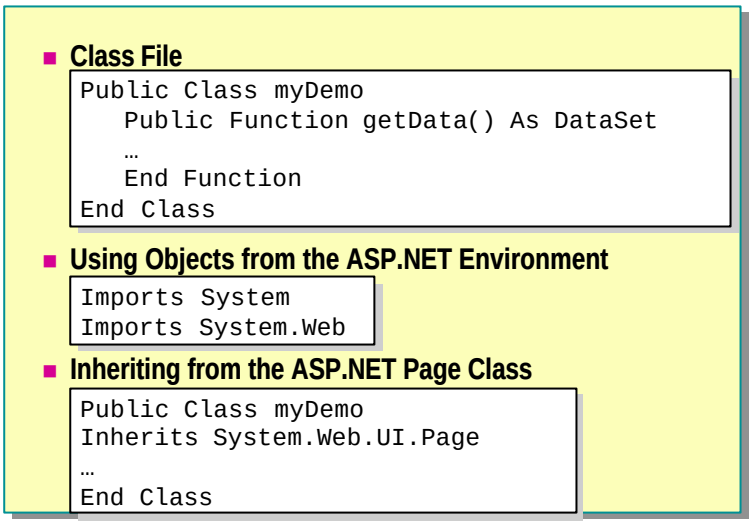
In Visual Basic, the class file has a .vb extension. In Visual C#, the code-behind page has a .cs extension. This ensures that the code-behind file is passed to the correct compiler when the page is first executed.

Using Code-Behind Procedures

To call a function or sub-procedure in a code-behind page, preface the name of the procedure with the name of the code-behind page. For example, in the class **myDemo**, in the code-behind page demo.vb, a function named **getData** returns a **DataSet** object. The following sample code calls the **getData** function and then displays the returned **DataSet** (ds) in the **DataGrid** control (dgAuthors).

```
Public Sub Page_Load (Src As Object, E As EventArgs)
   Dim ds As DataSet
   ds = getData()

   dgAuthors.DataSource=ds
   dgAuthors.DataMember="Authors"
   dgAuthors.DataBind()
End Sub
```

Creating a Class File in Visual Basic



A code-behind class file is identical to any other class file that you might create in your chosen programming language. The basic structure of a Visual Basic class file looks like the following:

```
Public Class class_name
  Public variable_name As variable_type
  Public Function function_name(parameters) As return_type
  …
  End Function
  Public Sub sub_name(parameters)
  …
  End Sub
End Class
```

Note Notice that, in the above class declaration, the functions, sub-procedures, and the class are all public. Public procedures are used to invoke code-behind pages from the user interface page.

To convert the preceding class file structure into a code-behind class file, you need to perform two steps:

1. Use the objects from the ASP.NET environment.

   You need to import the objects from the ASP.NET environment that you need to use in a class file. By default, all objects in an ASPX page import the objects from the ASP.NET environment. However, this is not true for a class file. Therefore, a class file, at the minimum, needs to import the System and Web libraries from the ASP.NET environment. This is done as follows:

   ```
   Imports System
   Imports System.Web
   ```

2. Inherit from the ASP.NET Page class.

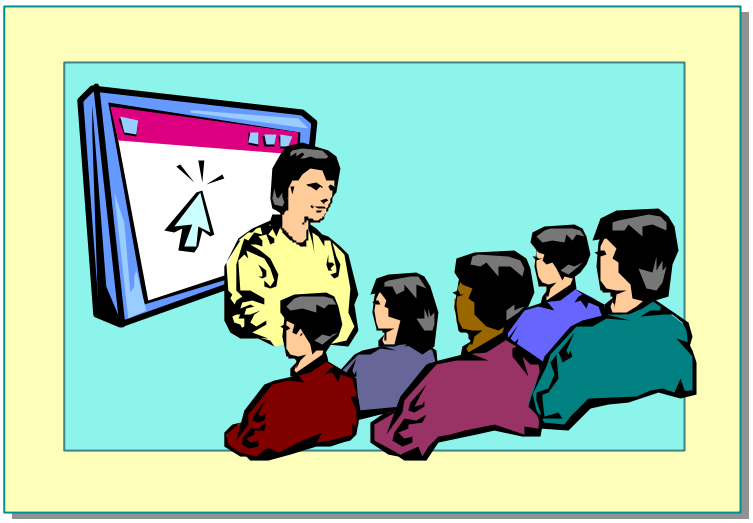
   Your class must inherit from the ASP.NET Page class so that it can be integrated into the ASP.NET page in which it is used. This is done with the **Inherits** statement.

   ```
   Public Class class_name
   Inherits System.Web.UI.Page
   …
   End Class
   ```

The following is a code-behind page with one class named **myDemo,** and one public function named **getData**. The function reads data from a database and returns it to the user interface page as a **DataSet** object.

```
Imports System
Imports System.Web
Imports System.Data
Imports System.Data.SqlClient

Public Class myDemo
Inherits System.Web.UI.Page

Public Function getData() As DataSet
   Dim ds As DataSet
   Dim conn As SqlConnection
   Dim cmdAuthors As SqlDataAdapter

   conn = New SqlConnection _
      ("server=localhost;uid=sa;pwd=;database=pubs")
   cmdAuthors = New SqlDataAdapter _
      ("select * from Authors", conn)

   ds = new DataSet()
   cmdAuthors.Fill(ds, "Authors")

   return (ds)
End Function
End Class
```

Demonstration: Creating a Code-Behind Page



In this demonstration, you will see how to create a code-behind page and use it from an ASP.NET page.

↙ To run the demonstration

1. Edit the page *<install folder>*\Democode\Mod04\beforeCodeBehind.aspx using Notepad.

   This page has both code and content.

2. Open the files codeBehindDemo.vb and codeBehind.aspx in the folder *<install folder>*\Democode\Mod04.

   Important The file codebehind.aspx cannot be opened in Visual Studio .NET. You must use Notepad or another text editor.

   These files perform the same function as the beforeCodeBehind.aspx page, but use a code-behind page.

3. View the page codeBehind.aspx in Microsoft Internet Explorer.

↙ To demonstrate using code-behind pages in Microsoft Visual Studio® .NET

Visual Studio .NET uses code-behind pages by default.

1. Add a new Web form to a Visual Studio .NET project.
2. Add a label Web control and a button Web control from the toolbox to the page.
3. View the HTML for the page and notice the attributes of the @Page directive.
4. Return to **Design** view and double-click the button Web control.

   This opens the code-behind page, and creates a click event procedure for the button.

   ```
   Public Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
   End Sub
   ```

5. Add code to change the **Text** property of the label control.

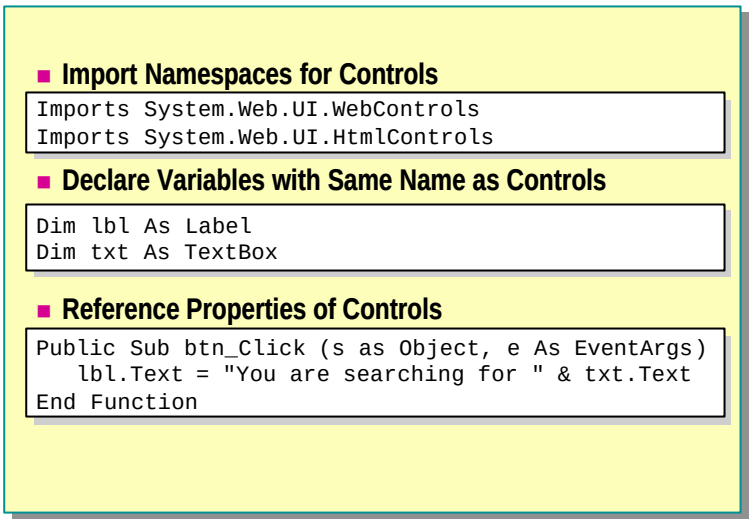
   ```
   Label1.Text = "clicked"
   ```

6. Build the project.

   In Visual Studio .NET, the code-behind pages are compiled into an assembly. You need to build the project before they will be accessible.

7. View the page in Internet Explorer.

Accessing Controls in a Code-Behind Page



You can create event procedures for the controls on a form in a code-behind page. In this way, you can transfer most of the code from the user interface page to a code-behind page.

To do this, you need to import the namespace for the controls you are using, either System.Web.UI.WebControls or System.Web.UI.HtmlControls. You then need to declare public variables for the controls on the ASPX page.

For example, the following is a form on the ASPX page with three controls:

```
<form runat=server>
  <asp:textbox id="txt" runat="server" />
  <asp:button id="btn" onclick="btn_click"
      text="Click Me" runat="server"/>
  <P><asp:label id="lbl" runat="server"/>
</form>
```

The code-behind page can include the event procedure and reference the controls on the page, as shown in the following example:

```
Imports System
Imports System.Web
Imports System.Web.UI.WebControls

Public Class myDemo
Inherits System.Web.UI.Page

  Public lbl As Label
  Public txt As TextBox

  Public Sub btn_click(s As Object, e As EventArgs)
      lbl.Text = txt.Text
  End sub
End Class
```

◆ Creating and Using User Controls

- Creating a User Control
- Using a User Control in an ASP.NET Page
- Demonstration: Creating a User Control

A user control is an ASP.NET page that is imported as a server control by another ASP.NET page. User controls provide an easy way to partition and reuse simple and common user interface (UI) functionality across a Web application. User controls are not pre-compiled. However, because all ASP.NET pages are compiled as soon as they are requested, user controls are compiled on demand and cached in server memory.

User controls have many advantages:

- User controls are self-contained. They provide separate variable namespaces, which means that none of the methods and variables of the user control conflict with any existing methods and variables of the hosting page.
- User controls can be used more than once within a hosting page without causing variable and method conflicts.
- User controls can be written in a different language from the main hosting page.

The main difference between code-behind pages and user controls is that code-behind pages mainly involve inheriting code classes into a page and user controls also allow you to generate parts of the user interface.

Creating a User Control

- **User-Defined Server Control with an .ascx Extension**
  - <%@ Control Language="vb" %>
  - Contains HTML but not <HTML>, <BODY>, or <FORM>
  - Contains code to handle its own events
- **Public Properties and Methods are Promoted to Properties and Methods of the Control in the Host Page**

```
Public Property pNum As Integer
  Get
      Return CInt(txtNum.Text)
  End Get
End Property
```

User controls are ASP.NET pages used as server controls. User controls have an .ascx extension. This file extension ensures that the user control's page cannot be executed as a stand-alone ASP.NET page. User controls largely replace include files. They are used to address the same problems that were solved by using include files, such as dealing with headers, navigation bar, repeating blocks of code, and so on.

A user control consists of HTML and code, but because user controls are included in existing pages, they do not contain <head>, <body>, or <form> tags. Those tags are included in the host ASP.NET page. User controls participate in the complete execution life cycle of every request and can handle their own events, encapsulating some of the page logic from the containing ASP.NET page. For example, a user control can handle its own postback in its **Page_Load** event procedure.

The following code is the HTML part of a user control that combines a text box and two input validation controls:

```
<%@ Control Language="vb" %>

<asp:textbox id="txtNum" runat="server" />
<asp:RequiredFieldValidator id="txtNumValidator"
      runat="server"
      controlToValidate="txtNum"
      errorMessage="You must enter a value"
      display="dynamic">
</asp:RequiredFieldValidator>
<asp:RangeValidator id="txtNumRngValidator" runat="server"
      controlToValidate="txtNum"
      errorMessage="Please enter a number between 0 and 99"
      type="Integer"
      minimumValue="0"
      maximumValue="99"
      display="dynamic">
</asp:RangeValidator>
```

To expose values of controls to the containing page, create public properties. For example, the following code creates a property named **pNum** that is the **Text** property of the text box control in the user control:

```
<SCRIPT language="VB" runat="Server">
Public Property pNum As Integer
  Get
      Return CInt(txtNum.Text)
  End Get
  Set
      txtNum.Text = Value.ToString()
  End Set
End Property
</SCRIPT>
```

All public variables, properties, and methods of a user control are promoted to properties (that is, tag attributes) and methods of the control in the containing page.

@Control directive

You can also have code-behind pages for user controls. The **@Control** directive is used to reference a user control from a code-behind page. It also defines user-control-specific attributes used by the ASP.NET page parser and compiler. This directive can only be used with user controls.

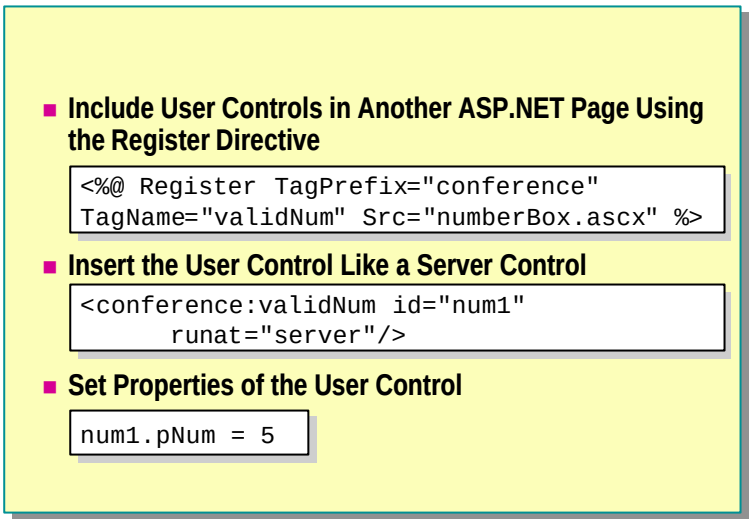
```
<%@ Control attribute=value [attribute=value … ]%>
```

For example, to reference a code-behind page for a user control, you use the following line of code:

```
<%@ Control Inherits="myDemo" src="demo.vb"%>
```

Note The **@Control** directive supports the same attributes as the **@Page** directive, except the **AspCompat** and **Trace** attributes. To enable tracing, you must add the **Trace** attribute to an **@Page** directive in the ASPX page that contains the user control. You can only include one **@Control** directive per .ascx file.

Using a User Control in an ASP.NET Page



User controls are included in an ASP.NET page by using the **@Register** directive:

```
<%@ Register TagPrefix="conference" TagName="validnum"
  Src="numberbox.ascx" %>
```

The **TagPrefix** attribute determines a unique namespace for the user control so that multiple user controls with the same name can be differentiated from each other. The **TagName** attribute is the unique name for the user control. The **Src** attribute is the virtual path to the user control file. After registering the user control with the **@Register** directive, you can place the user control tag in the ASP.NET page just as you would an ordinary server control (including the runat="server" attribute).

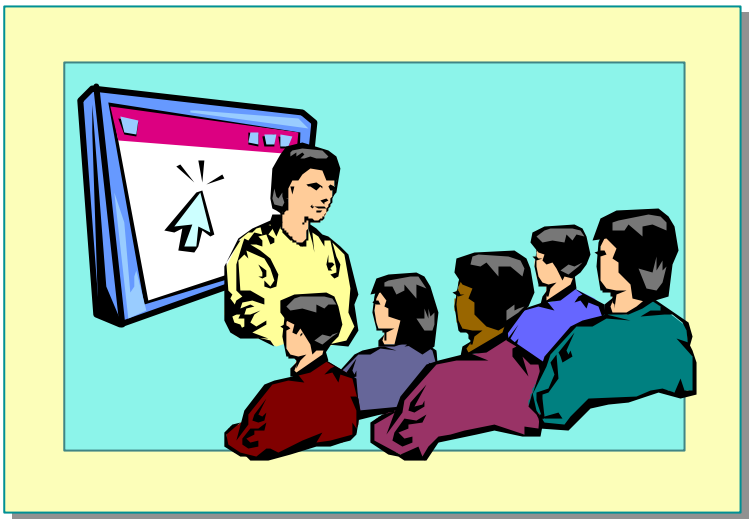
```
<conference:validNum id="num1" runat="server"/>
<conference:validNum id="num2" runat="server"/>
```

When the main ASP.NET page is loaded, the run-time framework compiles the user control file and makes it available to the page.

After the user control is available to the page, you can access its properties as follows:

```
lblSum.Text = (num1.pNum + num2.pNum).ToString()
```

Demonstration: Creating a User Control



In this demonstration, you will see how to create and use a user control.

The files numberbox.ascx and add.aspx in the folder *<install folder>*\Democode\Mod04 contain the completed code for this demonstration.

To run the demonstration

1. Open the page *<install folder>*\Democode\Mod04\beforeuser.aspx.

   This page has all controls and logic in one file.

2. Open the page numberbox.ascx in the folder *<install folder>*\Democode\Mod04.

   Numberbox.ascx is a user control that performs the same function as each text box on the page beforeuser.asps.

3. Add the **@Register** directive to the beforeuser.aspx page to use the numberbox.ascx user control.

```
<%@ Register TagPrefix="demos" TagName="validNum"
Src="numberbox.ascx" %>
```

4. Delete the four input validation controls from the beforeuser.aspx page.

5. Change the two text box controls to be <demos:validNum> controls. Use the same ID values.

```
Num1: <demos:validNum id="txtNum1" runat="server" />
Num2: <demos:validNum id="txtNum2" runat="server" />
```

6. Change the **Submit** event procedure to read values from the user controls.

```
Sub submit(s As Object, e As EventArgs)
   If Page.IsValid Then
       lblSum.Text = _
           CStr(Convert.ToInt32(txtNum1.pNum) + _
                Convert.ToInt32(txtNum2.pNum))
   End If
End Sub
```

7. View the beforeuser.aspx page in Internet Explorer.
8. View the HTML source in Internet Explorer.

◆ Creating and Using Components

- Deploying Components
- Creating Components
- Using Components in an ASP.NET Page
- Demonstration: Creating and Using a Component

In ASP.NET, business objects are the building blocks for multi-tiered Web applications, such as those with a layer for data access or common application rules. In this section, you will learn how to write some simple components and include them in your ASP.NET pages.

The classic use for an external component is in a two-tier scenario to perform data access. This simplifies the code in your page, improving readability and separating your UI logic from the back-end functionality.

The three-tiered application model extends the two-tiered scenario to include business rules between the UI and data access logic. The three-tiered model allows UI developers to work with a higher level of abstraction rather than directly manipulating data through low-level data access component application programming interfaces (APIs). The middle business component typically enforces business rules and ensures that the relationships and primary key restraints of the database are enforced.

Deploying Components

- **No Registration Required**
  - Components may be deployed by simply copying to the /bin dir or performing an FTP file transfer
- **No Server Restart Required**
  - The Web server does not require a restart when you change your application
  - New requests immediately begin execution against the changed file
- **No Namespace Conflicts**
  - Each assembly loaded from /bin is limited in scope to the application in which it is running

---

A problem with using the classic Component Object Model (COM) model for Web application components is that the components must be registered on the server before they can be used from a traditional ASP application. Remote administration of these components is often not possible, because the registration utility must be run locally on the server. In addition, these components remain locked on disk after they are loaded by an application and the entire Web server must be stopped before these components can be replaced or removed.

ASP.NET solves these problems by allowing components to be placed in a well-known directory that is automatically found at run time. This directory is named /bin, and is located immediately under the root directory for the application.

Following are the advantages of storing components in the /bin directory:

- No registration required

  No registration is required to make components available to the ASP.NET application. Components may be deployed by simply copying a file to the /bin directory or performing a File Transfer Protocol (FTP) file transfer.

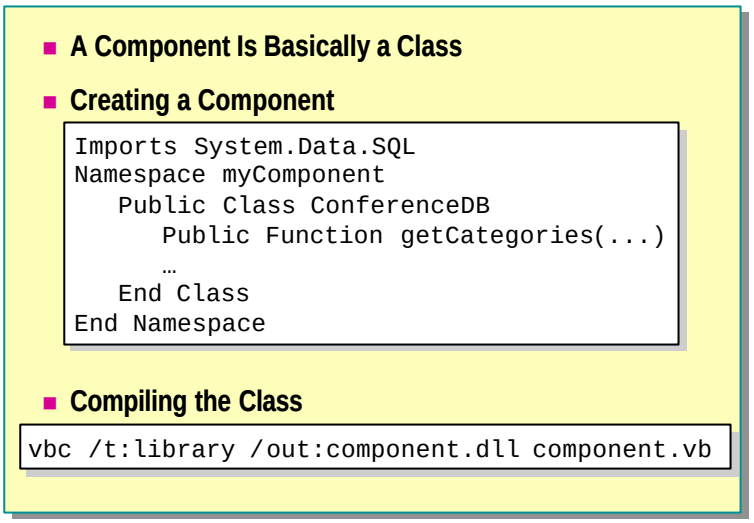
- No server restart required

  When the component is changed by replacing a .dll in the /bin directory, new requests by ASP.NET applications immediately begin execution against the changed file(s). ASP.NET allows currently executing requests to complete before the old application is replaced with the updated component(s). The Web server does not require a restart when you change components.

- No namespace conflicts

  Each component loaded from /bin is limited in scope to the application in which it is running. This means that many applications could potentially use different components with the same class or namespace names, without conflict.

Creating Components



A component at its most basic level is simply a class. The component can be instantiated from an ASP.NET page that imports the component.

Code-Behind vs. Components

A code-behind page is an uncompiled class file. It does not define a namespace that can be used by many applications. Instead, it defines a class that the .aspx page inherits from. Therefore, the .aspx page can immediately call methods and use properties of the code-behind page; you do not need to instantiate an object. An .aspx page uses a code-behind page by adding the following to the @Page directive:

Code-Behind Example

```
<%@ Page Language="vb" Inherits="myDemo" Src="demo.vb" %>
```

A component is always compiled into a dynamic-link library (DLL). Any page can access the DLL. You need to instantiate an object of the class before calling its methods. An ASPX page uses a component by adding the following code:

Component Example

```
<%@ Import Namespace="myComponent" %>
```

Creating a Component

A component class file includes a namespace declaration and class declarations.

```
Namespace myComponent
  Public Class ConferenceDB
  . . .
  End Class
End Namespace
```

Namespaces are used as an organizational system. They provide a way to present program components that are exposed to other programs and applications. Namespaces are always Public; however, the classes within the namespace may have Public, Friend, or Private access.

Just as you would with code-behind pages, you need to import any libraries that you use in the page.

```
Imports System.Data
Imports System.Data.SqlClient
```

Example

The following code is an example of a simple class called the **ConferenceDB** class. This class has a function that retrieves a list of categories from the database.

```
Imports System.Data
Imports System.Data.SqlClient

Namespace myComponent
Public Class ConferenceDB

  Public Function getCategories _
      (conn As SqlConnection) As DataSet
      'Retrieve a list of categories from the database
  End Function

End Class
End Namespace
```

Compiling the Component

To compile this class, you can either click **Build** on the **Build** menu in Visual Studio .NET or run the compiler from the MS-DOS® command prompt. The **/t** option tells the compiler that you want to build a library (.dll), and the **/out** option tells the compiler where to place the resulting assembly. An assembly is a collection of resources that are built to work together to deliver a cohesive set of functionality. An assembly carries all the rules necessary to ensure cohesion of functionalities. It is the unit of access to the resources in the Common Language Runtime (CLR).

```
vbc /t:library /out:component.dll component.vb
```

If the component imports other libraries, you must reference them in the compile statement. For example, if you are using the System.Data namespace, include the System.Data.dll library, as shown in the following example:

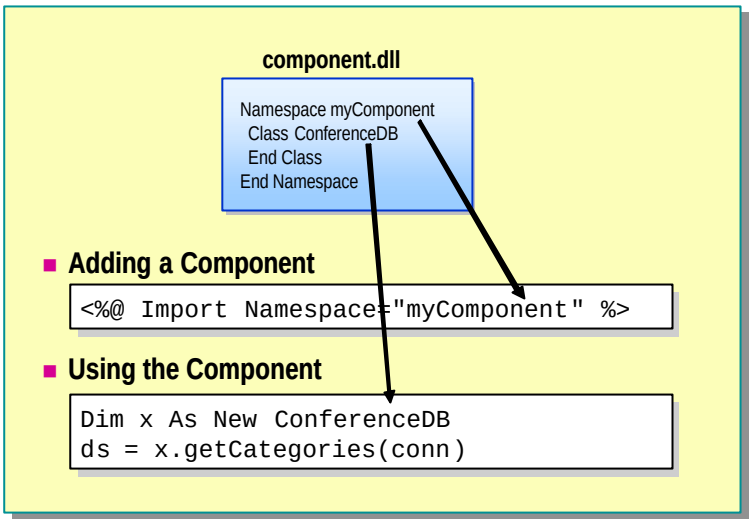
```
vbc /t:library /out:component.dll /r:System.Data.dll
component.vb
```

Finally, after compiling the component, move it to the \bin folder in the root of the virtual directory for your Web site.

Important You must define a virtual directory for your Web site.

The component is now available for use in an ASP.NET page.

Using Components in an ASP.NET Page



By default, ASP.NET loads all assemblies from the /bin directory when the application is started. The assemblies to load are specified through the configuration system. Additional assemblies may also be imported into an application by using the configuration file.

Because the assembly is pre-loaded by the ASP.NET run time, only a simple namespace import or variable declaration is required to make the component available.

There are two ways to use the classes defined in a component. You can either import the namespace by using an **@Import** directive, or reference the namespace when declaring a variable.

Using the @Import directive

To include any component in an ASP.NET page, you need to add the **@Import** directive that specifies the namespace to include at the top of the page. For example, to include the **myComponent** component, you add the following directive at the top of the page:

```
<%@ Import Namespace="myComponent" %>
```

You can then instantiate an object of the class by referencing the class name directly, as shown in the following example.

```
Dim x As New ConferenceDB
```

Referencing Namespace when Declaring a Variable

You can also declare a variable directly from a component, without adding the **@Import** directive to the page, by referencing the namespace in the declaration. For example, to declare a variable from the **myComponent** component, use the following code:

```
Dim x As New myComponent.ConferenceDB
```

Note If you create and compile the component in Visual Studio .NET, you will need to preface the namespace with the name of the Visual Studio .NET project. For example, if the component.vb file is located in a Visual Studio .NET project named demos, you would use the following **Imports** statement or variable declaration to reference it:

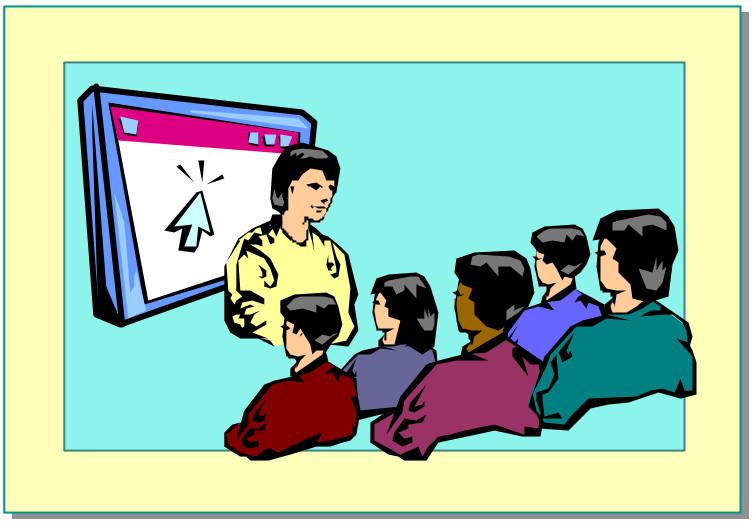
```
<%@ Import Namespace="demos.myComponent" %>
or
Dim x As New demos.myComponent.ConferenceDB
```

Calling Component Methods

After you have instantiated an object from the class, you can access its properties and methods. The following example shows how to call the **getCategories** method of the **ConferenceDB** class.

```
ds = x.getCategories(conn)
```

Demonstration: Creating and Using a Component



In this demonstration, you will see a simple component and how to use it in an ASP.NET page.

↙ To run the demonstration

1. Open the file component.vb from the folder <*install folder*>\DemoCode\Mod04.
2. Compile the file (manually, or using Visual Studio .NET).
   a. If you use the batch file mkdemos.bat to compile the component, you need to create a \bin folder in the <*install folder*> folder, which is the root of the 2063 virtual directory, and move the component.dll file into it.
   b. If you build in Visual Studio .NET, the assembly is automatically moved to the \bin folder of the project, but you will need to change the **@Import** directive in the displaycomponent.aspx page to:

      ```
      <%@ Import Namespace="projectname.myComponent" %>
      ```

3. Open the displaycomponent.aspx page.

   The **@Import** directive imports the namespace from the component. The **getCategories** function is called in the **Page_Load** event procedure.
4. View the displaycomponent.aspx page in Internet Explorer. View the source HTML of the page.

   The code from the component was not added to the file.

Lab 4: Separating Code from Content

Objectives

After completing this lab, you will be able to:

- Create a user control.
- Insert a user control into an .aspx page.
- Create and use components.

Prerequisite

Before working on this lab, you must know how to use Web controls in an ASP.NET page.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab04\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab04\Solution.

The ASPNET Web application uses classes in the Conference component. There are five files in this component: ConferenceDB.vb, CustomersDB.vb, ProductsDB.vb, ShoppingCartDB.vb, and OrdersDB.vb. These files are in the Components folder of the ASPNET Web application. If, for any reason, you need to rebuild the Conference component, run the mk.bat batch file from a command prompt. The mk.bat batch file creates the conference.dll component and then puts it in the bin folder of the ASPNET Web application.

Scenario

All the Web pages in the ASPNET Web site have the same controls at the top and down the left side of the page. These controls can be put into user controls and then easily added to the pages. In Exercise 1, you will create a user control and in Exercise 2, you will add the user control to the ProductsList.aspx page.

Also, many of the pages in the ASPNET Web site use data obtained from the Conference database. In Exercise 3, you will create a component to isolate and reuse the data access code. In Exercise 4, you will call methods on your component from the ProductsList.aspx page.

Estimated time to complete this lab: 60 minutes

Exercise 1
Creating a Menu User Control

In this exercise, you will convert an .aspx page into a user control. The page Menu.aspx reads categories of data from the database and displays them in a **DataList** Web control. In this exercise, you will convert the Menu.aspx page into a user control.

↙ To create the user control file

1. Open the file named Menu.aspx from the ASPNET Web site.
2. Remove the HTML tags.
3. Remove the BODY tags.
4. Change the @Page directive into an @Control directive.

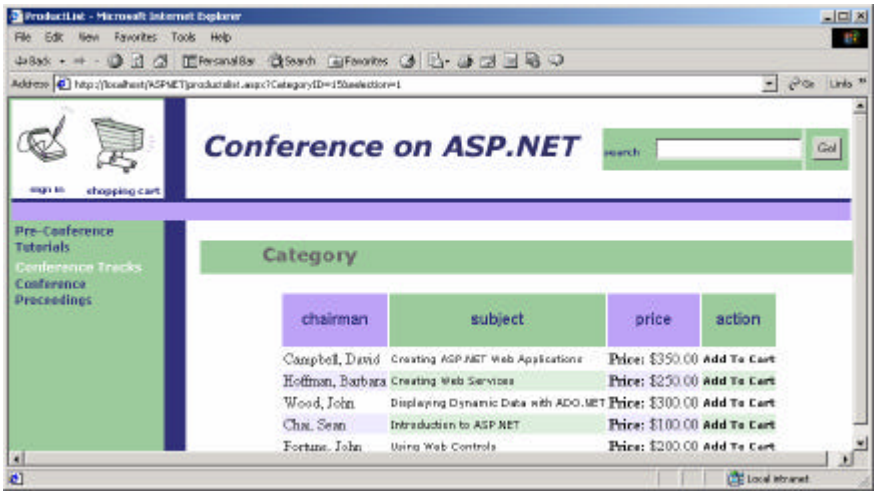
```
<%@ Control Language="vb" %>
```

5. Save the page as _Menu.ascx (overwrite the existing _Menu.ascx file).

Exercise 2
Using User Controls

In this exercise, you will add two user controls to the ProductsList.aspx page. One control will be located along the top of the page and can be found in the file _header.ascx. The other will be located at the left side of the page and can be found in the file _menu.ascx that you just created in Exercise 1.

When you are finished with this exercise, the page ProductsList.aspx should look like the following illustration.



↙ To add the header and menu user controls

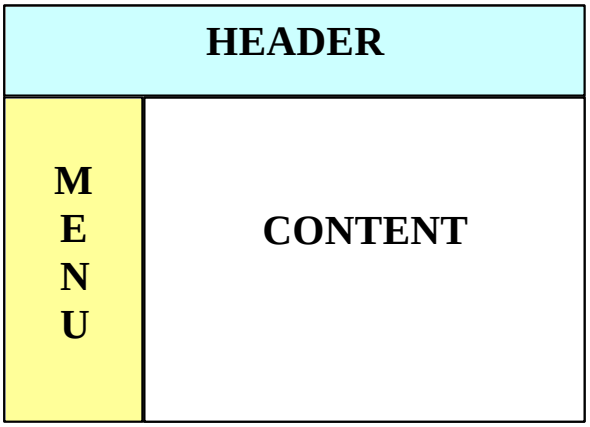
1. Open the page ProductsList.aspx in the ASPNET Web site.
2. Include the header and menu user controls at the beginning of the page by using the **@Register** directive. Set the attributes for the controls as shown in the following table.

TagPrefix	TagName	Src
Conference	Header	_Header.ascx
Conference	Menu	_Menu.ascx

Your code should look like the following:

```
<%@ Register TagPrefix="Conference" TagName="Header"
   Src="_Header.ascx" %>
<%@ Register TagPrefix="Conference" TagName="Menu"
   Src="_Menu.ascx" %>
```

3. Create an HTML table in the body of the page to hold the two user controls and the contents of the original page. You will need to build the table around the existing contents of the page.



Your HTML code should look like the following:

```
<table width="100%" border=0>
   <tr><td>
       <!-- _Header.ascx user control -->
   </td></tr>
   <tr><td>
       <table border=0>
           <tr><td valign="top">
               <!-- _Menu.ascx user control -->
               </td>
               <td>

                   <!-- Existing contents of the page -->
                   <!-- between <body> and </body> tags -->

               </td>
           </tr>
       </table>
   </td></tr>
</table>
```

4. Add the header and menu user controls to the correct position in the table.

   Your HTML code should look like the following:

```
<table width="100%" border=0>
   <tr><td>
       <!-- _Header.ascx user control -->
       <Conference:Header runat="server"/>
   </td></tr>
   <tr><td>
       <table border=0>
           <tr><td valign="top">
               <!-- _Menu.ascx user control -->
               <Conference:Menu runat="server"/>
               </td>
               <td>

                   <!-- Existing contents of the page -->
                   <!-- between <body> and </body> tags -->

               </td>
           </tr>
       </table>
   </td></tr>
</table>
```

↙ To save and test your work

1. Save your changes to ProductsList.aspx.
2. Using Internet Explorer, go to the ProductsList page of the ASPNET Web site by viewing:
   http://localhost/ASPNET/ProductsList.aspx?CategoryID=15

Note CategoryID=15 corresponds to the **Conference Tracks**category in the **Categories** table of the database.

Notice the header and menu user controls.

Exercise 3
Building a Component

Currently, the page ProductsList.aspx reads data from the database and displays it in a **Repeater** control. In this exercise, you will build a component, GetProducts.vb, to read the data from the database. Then, in Exercise 4, you will use that component in the ProductsList.aspx page.

↙ To create the GetProducts.vb component

1. Create a new file named GetProducts.vb in the **Components** folder of the ASPNET Web site.

   If you are using Visual Studio .NET, do the following:

   a. Right-click on the **Components** folder in the **ASPNET** project in the **Solution Explorer** window, click **Add**, and then click **Add Component**.

   b. In the **Add New Item** dialog box, click **Component Class**, type **GetProducts.vb** in the **Name** field, and then click **Open** to generate and open the new file.

2. Add the following code to import namespaces:

```
Imports System.Data
Imports System.Data.SqlClient
```

3. Create a new namespace named **db2063**.

4. Create a public class named **Products**.

   Your code should look like the following:

```
Namespace db2063
Public Class Products

End Class
End Namespace
```

↙ To add a method to the Products class

1. Create a new function named **GetProducts** that takes an Integer parameter named **categoryID** and returns a **DataSet**.

```
Public Function GetProducts(categoryID As Integer) As _
   DataSet
```

2. Open the ProductsList.aspx page.
3. Cut the content of the **Page_Load** event procedure, except for the first instruction that reads the **CategoryID** query string parameter and the last two instructions that set the **DataSource** of the **Repeater** controls. The **Page_Load** event procedure code should look like the following after you have cut the instructions as specified above:

```
Sub Page_Load(Source As Object, E As EventArgs)
   'Obtain CategoryID from QueryString
   Dim categoryID As Integer = _
        CInt(Request.Params("CategoryID"))

   ' Bind to the control
   repList.DataSource = dsProducts.Tables(0).DefaultView
   repList.DataBind()
End Sub
```

4. Save the changes to ProductsList.aspx.
5. Paste this code into the **GetProducts** function in the GetProducts.vb component.

6. Add a return statement to return the **Dataset** at the end of the function.

   Your function should look like the following:

```
Public Function GetProducts(categoryID As Integer) As _
DataSet
    Dim conn              As SqlConnection
    Dim strConn           As String
    Dim cmdProducts       As SqlDataAdapter
    Dim paramCategoryID   As SqlParameter
    Dim dsProducts        As New DataSet

    ' Create the connection
    strConn = _
        "server=localhost;uid=sa;pwd=1Aspnet;database=Conf"
    conn = New SqlConnection(strConn)

    ' Call the ProductsByCategory stored procedure
    cmdProducts = _
        New SqlDataAdapter("ProductsByCategory", conn)
    cmdProducts.SelectCommand.CommandType = _
        CommandType.StoredProcedure

    ' Add Parameters
    paramCategoryID = _
        New SqlParameter("@CategoryID", SqlDbType.Int, 4)
    paramCategoryID.Value = categoryID
    cmdProducts.SelectCommand.Parameters.Add(paramCategoryID)

    ' Fill the DataSet
    cmdProducts.Fill(dsProducts, "Products")

    ' Return the DataSet
    return dsProducts

End Function
```

Note Visual Studio .NET Beta 2 may insert parentheses at the end of your function declaration or elsewhere. Be sure to compare your code with the preceding example and remove any extra parentheses before continuing.

---

7. Save your changes to GetProducts.vb.

↙ To compile the component

There are two ways to compile the component: in an MS-DOS command prompt, or in Visual Studio .NET.

1. To compile the component in an MS-DOS command prompt:
   a. Open an MS-DOS command prompt.
   b. Navigate to the folder InetPub\wwwroot\ASPNET\components.
   c. Run the following command:

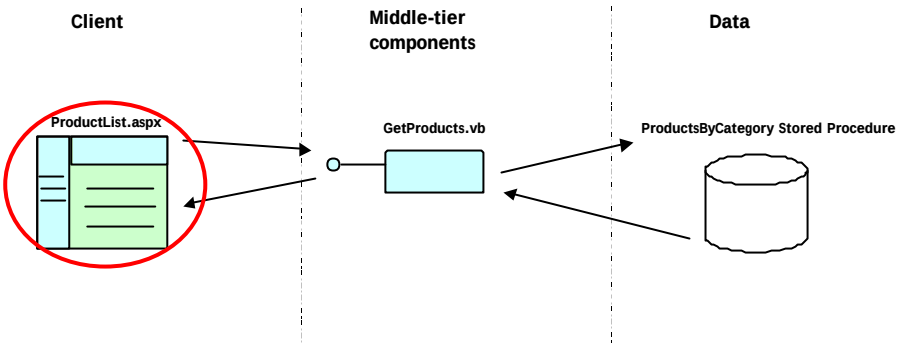
      ```
      vbc /t:library /out:..\bin\db2063.dll /r:System.dll
      /r:System.Web.dll /r:System.Xml.dll /r:System.Data.dll
      GetProducts.vb
      ```

      The compiler generates a file named db2063.dll in the ASPNET\bin directory.
2. To compile the component in Visual Studio .NET, on the **Build** menu, click **Build**.

   The compiler adds the new class to the project assembly named ASPNET.dll in the ASPNET\bin directory.

Exercise 4
Calling a Component

In this exercise, you will call the component that you built in Exercise 3 to get a **DataSet** of products that you will display on the ProductsList.aspx page.



↙ To call the component from the ProductsList.aspx page

1. Open the page ProductsList.aspx from the ASPNET Web site.
2. Edit the **Page_Load** event procedure.
3. Declare a variable named **productCatalog** that is an instance of the **Products** class in the db2063 namespace.

Note If you used Visual Studio .NET to compile the component, you must preface the namespace of the component with the root namespace (the project name) of the project.

If you used a command prompt to compile the component, your code should look like the following:

```
Dim productCatalog As New db2063.Products
```

If you used Visual Studio .NET to compile the component, your code should look like the following:

```
Dim productCatalog As New ASPNET.db2063.Products
```

4. Declare a **DataSet** variable named dsProducts.
5. After the code that reads the **categoryID** parameter from the **QueryString** object, call the **GetProducts** method of the component and save the result in the dsProducts variable.

   Your code should look like the following:

```
Dim dsProducts As New DataSet

dsProducts = productCatalog.GetProducts(CategoryID)
```

↙ To save and test your work

1. Save your changes to ProductsList.aspx.
2. Using Internet Explorer, go to the ProductsList page of the ASPNET Web site by viewing:
   http://localhost/ASPNET/ProductsList.aspx?CategoryID=15

   The page should display the table of conference tracks.

Review

- Advantages of Partitioning an ASP.NET Page
- Creating and Using Code-Behind Pages
- Creating and Using User Controls
- Creating and Using Components

1. List some of the advantages of separating code from content.

2. How do you link the user interface page to its code-behind file?

3. What are the ASP.NET namespaces that you need to import in a code-behind class file?

4. What are some of the differences between a regular ASP.NET page and a user control?

5. What is the biggest advantage that ASP.NET offers for using external components?

6. How do you use a user control from an ASP.NET page?

7. How do you use a component from an ASP.NET page?